II. AMENDMENTS TO THE SPECIFICATION:

Please make the following amendments to the specification:

On page 11, para. 0027, please amend as follows:

As depicted, API system 32 includes input system 34, transformation system 36, class matching system 38, method listing system 40, class comparison system 42, removal system 44 and report generation system 46. It should be understood that each of these systems includes program code/logic for carrying out the functions described herein. Moreover, it should be understood that API system 32 could reside within an existing modeling tool such as Rational (as discussed above). To this extent, the present invention could be embodied as a modification to any modeling tool now know or later developed. Regardless, for the comparison to be performed under the present invention, the developer will provide some basic input. To this extent, as shown, the developer will first provide common class paths for uncompiled first release 14A and uncompiled second release 16A of Java code. Specifically, it could be the case that a class within the releases 14A and/or 16A references another class outside of the releases 14A and/or 16A. Providing the common class paths 54 ensures that such classes can be located. The developer will also provide source input 56 corresponding to uncompiled first release 14A and target input 58 corresponding to uncompiled second release 16A. Source input 56 and target input 58 can be a list of classes, one or more ~~jar~~ JAR files, or the like. In any event, the class paths 54, source input 56 and target input 58 will be received by input system 34 (and possibly stored in storage unit 30).

On pages 13-14, para. 0030, please amend as follows:

Referring to Figs. 2-4, flow diagrams are depicted to further describe the teachings of the present invention. Specifically, referring first to Fig. 2, the method as described above is depicted. As shown, first step S1 of method 100 is to receive common class paths. Second step S2 is to receive the source input corresponding to the first release of byte code and the target input corresponding to the second release of byte code. As indicated above, the input can be a list of classes, one or more ~~jar~~ JAR files or the like. In step S3, the source input and the target input are transformed into a first and second list of class names associated with the first and second releases of byte code, respectively. In step S4, the classes corresponding to any matching class names (i.e., class names that exist in both lists) are loaded. Once loaded the classes are compared in step S5.

On pages 14-15, para. 0033, please amend as follows:

It should be appreciated that the present invention need not be limited to the source input and target input being a list of classes. For example, referring to Fig. 4, a method 300 is depicted whereby the source input and the target input comprises one or more ~~jar~~ JAR files. Specifically, as shown, step J1 is to receive at least one source jar file and at least one target jar file. In step J2, the input jar files will be transformed into two lists of class names (as indicated in Fig. 2). In step J3, matching class names will be identified from the lists, and in step J4, the classes corresponding to the matching class names will be loaded. In step J5, the classes will be compared as described in conjunction with Fig. 3 After the comparison, the matching class name will be removed from the lists in step J6. Similar to method 100 of Fig. 2, the process will be

repeated for all matching class names. Once all matching class names have been removed from the lists, the process can end.